Lecture 9: Application of Cryptography

Lecture topics

- Cryptography basics
- Using SSL to secure communication links in J2EE programs
- Programmatic use of cryptography in Java

Cryptography basics

- Encryption
  - Transformation of data into a form that is almost impossible to read without the appropriate knowledge (key)
  - Decryption is the inverse transformation
- Authentication
  - Reliable determination of a specific fact
  - E.g. digital signatures authenticate authorship
  - E.g. digital timestamps authenticate creation time
Lecture 9: Application of Cryptography

**Perfect cryptography**

- Called one-time pad
- Create a key for each document anew, completely at random
- The length of the key == the length of the plaintext
- Apply the key to the plaintext (e.g. XOR) to obtain ciphertext
- An attacker cannot deduce the plaintext from the ciphertext, since the key is random
- Usually impractical
  - The key is long
  - Both parties need the key

**Private key cryptography**

- Also called secret key or symmetric cryptography
- A single key is used to both encrypt and decrypt messages
  - The key has to be known to both the sender and receiver
  - Block ciphers: give a secret key, transform a fixed-length block of plaintext (64 or 128 bits) into ciphertext of the same length
  - Stream ciphers: generates a keystream and applies it to the plaintext
- Challenge: have the sender and receiver agree on a key securely (key management problem)
Lecture 9: Application of Cryptography

**Public key cryptography**
- Diffie-Hellman invention (1976)
- Every party (corporation, person, program, etc) gets two keys: a public key and a private key
  - The public key is published (has to be associated with the party in a trusted manner)
  - The private key is kept secret
- The two keys in a pair are linked mathematically:
  - Plaintext encrypted with the public key can only be decrypted with the corresponding private key
  - Because of this relationship, it's possible to derive private key from the public key
    - Public key cryptosystems are designed in a way that makes the necessary computations prohibitively expensive (e.g. involving factoring large numbers)
- Can be used for digital signatures (authentication)
  - To sign a message, create a digital signature using the message and private key
  - If a computation involving the public key succeeds, signature has been verified

**Popular encryption algorithms**
- Private key
  - DES (Data Encryption Standard), ANSI standard
    - Designed by IBM, NSA, and NIST
    - 64 bits block size, 56-bit key
    - Currently triple-DES is the standard
  - AES (Advanced Encryption Standard)
    - Replaces DES; intended lifespan 20-30 years
    - Supports key sizes 128, 192, and 256 bits
- Public key
  - RSA (Rivest, Shamir, and Adleman) cryptosystem
    - Based on the assumption that factoring large integers is computationally hard
  - DSA (Digital Signature Algorithm)
    - Can be used only for digital signatures, not encryption
    - Based on the discrete logarithm problem
Lecture 9: Application of Cryptography

**Elliptic curve cryptography**

- Public key cryptosystems
- Proposed by Miller and Koblitz in mid-80s
- Based on operations over elliptic curves
  - Given two points $G$ and $Y$ on an elliptic curve such that $Y = kG$ (that is, $Y$ is $G$ added to itself $k$ times), find the integer $k$. This problem is commonly referred to as the elliptic curve discrete logarithm problem.

---

**Hash functions**

- Hash function is a transformation that takes an input and returns a fixed size string, called hash value
- Hash value is often called message digest
  - Useful for integrity checks and digital signatures
- Basic requirements for a hash function:
  - Input can be of any length
  - Output has a fixed length
  - $H(x)$ is relatively easy to compute for any $x$
  - $H(x)$ is one way ($H^{-1}$ is hard to compute)
  - $H(x)$ is collision free (computationally infeasible to find two strings $x$ and $y$ such that $H(x) = H(y)$)
Lecture 9: Application of Cryptography

**Popular hashing algorithms**

- **MD2, MD4, MD5 (Message Digest)**
  - Developed by Rivest
  - Intended for use in digital signatures
  - Produce 128-bit digest for an arbitrary length message
- **SHA and SHA-1 (Secure Hash Algorithm)**
  - Developed by NIST
  - Design similar to MD4
  - Produce 160-bit digest for a message of length no more than $2^{64}$ bits

**Public key certificates**

- A certificate is a digitally signed statement from a trusted entity (Certification Authority), saying what the public key of another entity is legit
- Sequence of steps involved:
  - An entity generates a key pair
  - The entity sends the public key to a CA (usually in email)
  - The CA generates the certificate as a binding between a X.500 hierarchical name identity and the public key
  - The CA signs the certificate and the public key with its own private key
- SSL handshake protocol for server authentication (client authentication is optional)
  - A client requests the server for its certificate
  - The server sends its certificate and cipher preferences
  - The client generates a master key and encrypts it with the public key of the server; sends to the server
  - The server decrypts the message and uses the master key to encrypt a message; sends to the client
What's in a certificate?

- The certificate issuer's name
- The entity for whom the certificate is being issued (aka the subject)
- The public key of the subject
- Some time stamps

So, how is a secure communication established (Alice wants to authenticate Bob, Bob has a pair of keys)

1. A -> are you Bob? -> B
2. B -> {are you Bob?}bobs-private-key -> A
3. A decrypts “are you Bob?” using bobs-public-key

Insufficient:
- Not a good idea to send around things signed with a private key --- can be used for impersonation
- Carl can get Bob to digitally sign something:
  1. C -> some data -> B
  2. B -> {some data}bobs-private-key -> C
  3. C -> {some data}bobs-private-key -> A
- Can use digests (hashes): Bob should create a digest from Alice’s message
  - Someone who tries to impersonate Bob cannot get the original message from the digest
Lecture 9: Application of Cryptography

So, how is a secure communication established (Alice wants to authenticate Bob, Bob has a pair of keys)

1. A \rightarrow \text{are you Bob?} \rightarrow B
2. B \rightarrow \text{Alice, this is Bob} \rightarrow A
3. B \rightarrow \text{\{digest[Alice, this is Bob]\}bobs-private-key} \rightarrow A
4. A decrypts the digest using bobs-public-key
5. A applies the hash function to ”Alice, this is Bob”, checks that the same digest is obtained

- Insufficient for handing out public keys
  1. A \rightarrow \text{are you Bob?} \rightarrow B
  2. B \rightarrow \text{Alice, this is Bob, bobs-public-key} \rightarrow A
  3. A \rightarrow \text{prove it} \rightarrow B
  4. B \rightarrow \text{\{digest[Alice, this is Bob]\}bobs-private-key} \rightarrow A

- Anybody can be Bob, all they need is to generate a pair of keys
- Need to use certificates

So, how is a secure communication established (Alice wants to authenticate Bob, Bob has a pair of keys)

1. A \rightarrow \text{are you Bob?} \rightarrow B
2. B \rightarrow \text{Alice, this is Bob, bobs-certificate} \rightarrow A
3. A \rightarrow \text{prove it} \rightarrow B
4. B \rightarrow \text{\{digest[Alice, this is Bob]\}bobs-private-key} \rightarrow A

- May be insufficient for sending secrets
  1. A \rightarrow \text{are you Bob?} \rightarrow B
  2. B \rightarrow \text{Alice, this is Bob, bobs-certificate} \rightarrow A
  3. A \rightarrow \text{prove it} \rightarrow B
  4. B \rightarrow \text{\{digest[Alice, this is Bob]\}bobs-private-key} \rightarrow A
  5. A \rightarrow \text{ok, here's a secret} \{\text{secret}\}bobs-public-key \rightarrow B
  6. B \rightarrow \text{some message} \text{secret} \rightarrow A

- May be susceptible to a parrot attack, where Carl sits between A and B; garbles messages encrypted by the secret key, hoping to get one in a right format
Lecture 9: Application of Cryptography

Should use Message Authentication Codes (MAC)

- MAC is computed as a digest of some piece of data and the secret:
  \[ \text{MAC} = \text{digest}\left[\text{some message, secret}\right] \]
- Carl’s attempts at garbling messages will have very little likelihood of success
  1. A \(\rightarrow\) are you Bob? \(\rightarrow\) B
  2. B \(\rightarrow\) Alice, this is Bob, bobs-certificate \(\rightarrow\) A
  3. A \(\rightarrow\) prove it \(\rightarrow\) B
  4. B \(\rightarrow\) \{digest[Alice, this is Bob]\}bobs-private-key \(\rightarrow\) A
  5. A \(\rightarrow\) ok, here’s a secret \{secret\}bobs-public-key \(\rightarrow\) B
  6. B \(\rightarrow\) \{some message, digest[some message, secret]\}secret \(\rightarrow\) A
- Finally, need to timestamp messages, to further protect against replay attacks

SSL – Secure Sockets Layer

- Network protocol for secure communications
  - Encryption of data
  - Authentication with digital certificates
- Widely used in Web commerce (https://…)
- SSL basically implements the protocol on the previous slide
Using SSL with EJB-based J2EE applications

- Would like to use SSL for sending data during remote calls to methods of EJBs
  - So that network sniffing does not hurt us
- Put the following in the jboss-service.xml configuration file:

```xml
<attribute name="RMIClientSocketFactory">
  org.jboss.security.ssl.RMIServerSocketFactory
</attribute>
<attribute name="RMIServerSocketFactory">
  org.jboss.security.ssl.RMIServerSocketFactory
</attribute>
<attribute name="SecurityDomain">java:/jaas/RMI+SSL</attribute>
</mbean>
```

Using SSL with EJB-based J2EE applications (cont.)

- Put the following in the jboss.xml file

```xml
<session>
  <ejb-name>MyBean</ejb-name>
  <configuration-name>Standard Stateful SessionBean</configuration-name>
  <home-invoker>jboss:service=invoker,type=jrmp,socketType=SSL</home-invoker>
  <bean-invoker>jboss:service=invoker,type=jrmp,socketType=SSL</bean-invoker>
</session>
```

- Put the following in the jbossmq-service.xml file (only if you use JMS)

```xml
<attribute name="SecurityDomain">java:/jaas/RMI+SSL</attribute>
```

Page <#>
Lecture 9: Application of Cryptography

**Pitfalls of using cryptography in your applications**
- Know when to use it
- Know when not to use it
- Know trade-offs involved
- Know infrastructure requirements
- Do not invent your own cryptographic algorithms
- Avoid coding existing cryptographic algorithms as much as possible
  - Good crypto libraries are available
  - Coding cryptography well is difficult

**Java Cryptography Architecture (JCA)**
- Introduced in JDK 1.1, significantly extended since
- Designed with flexibility in mind
  - Algorithm independence and extensibility
    - Often, digital signatures from different algorithms can be used interdependently
    - Engine classes: MessageDigest, Signature, KeyFactory, KeyPairGenerator, ... define cryptographic services in an abstract way (no implementation)
  - Implementation independence and interoperability
    - E.g., different implementations of an algorithm can deal with each other's keys
- Uses the concept of **Cryptographic Service Provider**
  - A set of packages that implement cryptographic services (digital signature and message digest algorithms, key conversion, digital certificates, ...)
  - "SUN" is the default provider
Example: MessageDigest engine

- Provides hashing services
- Can request specific algorithms (SHA-1 or MD5) from the engine
  - Factory method
    `MessageDigest.getInstance(algName);`
  - Optionally, can ask for a specific provider
    `MessageDigest.getInstance(algName, provider);`
- The digest method performs hashing
  - Byte array of arbitrary length is input
  - Byte array (20 bytes for SHA-1, 16 bytes for MD5) is output

Creating digital signatures

- Engine class Signature
  - Can request the algorithm and/or provider to use, e.g. RSA with MD5
  - Factory methods create Signature objects
- Signature objects are modal (always in one of several possible states):

  UNINITIALIZED
  - initSign(publicKey)
  - initVerify(privateKey)

  SIGN
  - initSign
  - update, sign, initSign

  VERIFY
  - initVerify
  - update, verify, initVerify
Lecture 9: Application of Cryptography

Configuring providers

- Multiple cryptographic providers can be installed and used at the same time
- A provider may supply only some crypto services
- Providers can be given the order of preference
  - E.g. “for DES implementation, use the IBM provider; if that provider does not provide an implementation, use SUN”
- If no provider has an implementation, NoSuchAlgorithmException is thrown
- Installing providers
  - Usually distributed as JAR files
  - Let the JVM know about providers and order of preference by modifying configuration file \($\text{JAVAHOME}/lib/security/java.security\)
  - E.g.
    - `security.provider.1=com.company.crypto.CompanyProvider`
    - `security.provider.5=com.anothercompany.pack.Prov`

In-class presentation of bookauction designs

- 15 minutes per group
  - As many people as you want can do the presentation
  - PowerPoint slides preferred
- Don’t have to describe everything
  - Concentrate on the changes to my design
  - What are the entity beans?
  - Do you have session beans?
- What login procedures did you design?
- Give a URL where remote and home interfaces for your EJBS will be published